# Assignment 15

Design and implement a Python script to detect Deep Fake videos utilizing the "Deepfake Detection Challenge" dataset available on Kaggle.

1. Define the objective of the "Deepfake Detection Challenge" dataset.

2. Describe the characteristics of Deep Fake videos and the challenges associated with their detection.

3. Outline the key steps involved in the implementation of a Deep Fake video detection algorithm using Python.

4. Discuss the importance of dataset preprocessing in training a Deep Fake detection model and suggest potential preprocessing techniques.

5. Propose and justify the choice of at least two machine learning or deep learning algorithms suitable for Deep Fake video detection.

6. Evaluate the performance metrics that can be used to assess the effectiveness of a Deep Fake detection model.

7. Consider the ethical implications of Deep Fake technology and discuss the role of detection mechanisms in addressing these concerns.

8. Write a complete code for this assignment.

Recent years have seen a substantial increase in interest in deepfakes, a fast-developing field at the nexus of artificial intelligence and multimedia. These artificial media creations, made possible by deep learning algorithms, allow for the manipulation and creation of digital content that is extremely realistic and challenging to identify from authentic content. Deepfakes can be used for entertainment, education, and research; however, they pose a range of significant problems across various domains, such as misinformation, political manipulation, propaganda, reputational damage, and fraud.

Deepfakes are produced by manipulating existing videos and images to produce realistic-looking but wholly fake content. The rise of advanced artificial intelligence-based tools and software that require no technical expertise has made deepfake creation easier. With the unprecedented exponential advancement, the world is currently witnessing in generative artificial intelligence, the research community is in dire need of keeping informed on the most recent developments in deepfake generation and detection technologies to not fall behind in this critical arms race.

Deep fakes present a number of serious issues that arise in a variety of fields. These issues could significantly impact people, society, and the reliability of digital media. Some significant issues include fake news, which can lead to the propagation of deceptive information, manipulation of public opinion, and erosion of trust in media

sources. Deepfakes can also be employed as tools for political manipulation, influence elections, and destabilize public trust in political institutions. In addition, this technology enables malicious actors to create and distribute non-consensual explicit content to harass and cause reputational damage or create convincing impersonations of individuals, deceiving others for financial or personal gains. Furthermore, the rise of deep fakes poses a serious issue in the domain of digital forensics as it contributes to a general crisis of trust and authenticity in digital evidence used in litigation and criminal justice proceedings. All of these impacts show that deepfakes present a serious threat, especially in the current sensitive state of the international political climate and the high stakes at hand considering the conflicts on the global scene and how deepfakes and fake news can be weaponized in the ongoing media war, which can ultimately result in catastrophic consequences.

Therefore, deepfake detection techniques need to be constantly improved to catch up with the fast-paced evolution of generative artificial intelligence. There is a need for literature reviews to keep up with the fast-changing field of artificial intelligence and deepfakes to enable researchers and professionals to develop robust countermeasure methods and to lay the right groundwork to make it easier to detect and mitigate deepfakes.

Detecting video deepfakes using machine learning (ML) involves several steps, primarily focusing on analysing visual and audio components of the video. Here's a generalized process for detecting video deepfakes using ML:

**1. Data Collection and Pre-processing:**

— Gather a diverse dataset of both real and synthetic videos. This dataset should include examples of deepfake videos, as well as genuine recordings.

— Preprocess the video data, which may involve converting it into frames, resizing, and normalization.

**2. Feature Extraction:**

— Extract relevant features from the video frames that can be used to distinguish between real and fake videos. **Features might include:**

**— Facial landmarks and expressions**

**— Motion patterns and consistency**

**— Statistical features of frames and sequences**

**— Extract audio features such as spectrograms, pitch, and intensity.**

**3. Model Selection:**

— choose an appropriate machine learning model for video classification. Commonly used models include:

**— Convolutional Neural Networks (CNNs) for image analysis.**

**— Recurrent Neural Networks (RNNs) or 3D Convolutional Neural Networks (3D CNNs) for temporal data analysis.**

**— Hybrid architectures combining CNNs and RNNs for spatio-temporal analysis.**

**— Consider pre-trained models** or architectures specifically designed for video analysis tasks.

**4. Model Training:**

— **Split the dataset into training, validation, and test sets.**

**— Train the selected model on the training data using the extracted features from both video frames and audio tracks.**

**— Tune hyper parameters** such as learning rate, batch size, and model architecture to optimize performance.

**— Regularize the model to prevent over fitting by using techniques such as dropout, batch normalization, or early stopping.**

**5. Evaluation:**

— Evaluate the trained model on the validation set to assess its performance.

— **Use appropriate evaluation metrics for binary classification tasks, such as accuracy, precision, recall, F1 score, and receiver operating characteristic (ROC) curve analysis.**

— Adjust the model or training strategy based on validation performance.

**6. Testing and Deployment:**

— Evaluate the trained model on the test set to obtain unbiased performance estimates.

— Deploy the model in a real-world setting to identify video deepfakes.

— Integrate detection algorithms into automated content moderation systems to prevent the spread of deepfake videos online.

— Continuously monitor and update the model to adapt to new deepfake generation techniques and maintain effectiveness over time.

**7. Post-Deployment Monitoring and Maintenance:**

— Monitor the model's performance in detecting video deepfakes in real-world scenarios.

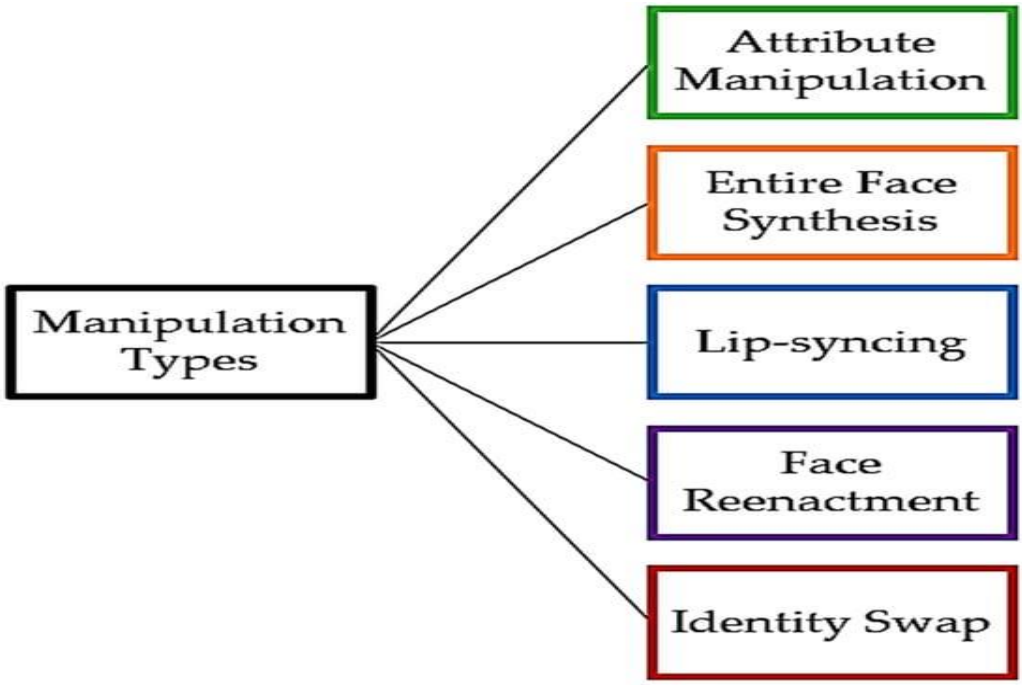
— Collect additional data if necessary to improve the model's robustness and generalization capabilities.

— Update the model periodically to incorporate new insights, techniques, or data.
By following this process, you can effectively detect video deepfakes using machine learning techniques, helping to mitigate the spread of disinformation and protect against the potential harms of deepfake technology.

## Deepfake Generation

Deepfake Manipulation Types

There exist five primary types of deepfake manipulation, as shown in **Figure**.




**attribute manipulation:** only the region that is relevant to the attribute is altered alone in order to change the facial appearance by removing or donning eyeglasses, retouching the skin, and even making some more significant changes, like changing the age and gender. Nevertheless, our attention is directed towards manipulations that are predominantly prevalent in video format due to their heightened engagement levels compared to image-based content. Consequently, it is more likely for people to fall victim to deepfake videos.

**Face synthesis: It** is a manipulation type which entails creating images of a human face that does not exist in real life

These manipulations are designed to make it appear as though a person is doing or saying something that they did not actually do or say.

**Face swapping:** it is a form of manipulation that has primarily become prevalent in videos even though it can occur at the image level. It entails the substitution of one individual's face in a video, known as the source, with the face of another person, referred to as the target. In this process, the original facial features and expressions of the target subject are mapped onto the associated areas of the source subject's face, creating a seamless integration of the target's appearance into the source video. The origins of research on the subject of identity swap can be traced to the morphing method introduced in

**Face reenactment:** it is a manipulation technique that focuses on altering the facial expressions of a person in a video. It involves the replacement of the original facial expression of the subject, with the facial expression of another person

**Lip-syncing:** where the objective is to generate a target face that appears authentic and synchronizes with given text or audio inputs. Achieving accurate lip movements and facial expressions that align with the source audio necessitates the use of advanced techniques.

Additionally, meticulous post-processing is crucial to ensuring that the resulting video portrays a natural and seamless facial appearance.

## Deepfake Generation Techniques

Multiple techniques exist for generating deepfakes. Generative Adversarial Networks GANs and Autoencoders are the most prevalent techniques. GANs consist of a pair of neural networks, a generator network and discriminator network, which engage in a competitive process. The generator network produces synthetic images, which are presented alongside real images to the discriminator network. The generator network learns to produce images that deceive the discriminator, while the discriminator network is trained to differentiate between real and synthetic images. Through iterative training, GANs become proficient at producing increasingly realistic deepfakes. On the other hand, Autoencoders can be used as feature extractors to encode and decode facial features. During training, the autoencoder learns to compress an input facial image into a lower-dimensional representation that retains essential facial features. This latent space representation can then be used to reconstruct the original image. Though, for deepfake generation, two autoencoders are leveraged, one trained on the face of the source and another trained on the target.

Numerous sophisticated GAN-based techniques have emerged in the literature, contributing to the advancement and complexity of deepfakes.

**AttGAN** is a technology for facial attribute manipulation; its attribute awareness enables precise and high-quality attribute changes, making it valuable for applications like face-swapping and age progression or regression. Likewise, **StyleGAN** is a GAN architecture that excels in generating highly realistic and detailed images. It allows for the manipulation of various facial features, making it a valuable tool for generating high-quality deepfakes. Similarly, **STGAN** modifies specific facial attributes in images while preserving the person's identity. The model can work with labelled and unlabelled data and has shown promising results in accurately controlling attribute changes. Another technique is **StarGANv2**, which is able to perform multi-domain image-to-image translation, enabling the generation of images across multiple different domains using a single unified model.

Unlike the original **StarGAN**, which could only perform one-to-one translation between each pair of domains, StarGANv2 can handle multiple domains simultaneously. An additional GAN variant is **CycleGAN**, which specializes in style transfer between two domains. It can be applied to transfer facial features from one individual to another, making it useful for face-swapping applications. Moreover, there is **RSGAN**, which can encode the appearances of faces and hair into underlying latent space representations, enabling the image appearances to be modified by manipulating the representations in the latent spaces. For a given audio input, **LipGAN** is intended to produce realistic lip motions and speech synchronization.

In addition to the previously mentioned methods, there is a range of open-source tools readily available for digital use, enabling users to create deep fakes with relative ease, like **FaceApp**, **Reface**, **DeepBrain**, **DeepFaceLab**, and **Deepfakes Web** .These tools have captured the public's attention due to their accessibility and ability to produce convincing deepfakes. It is essential for users to utilize these tools responsibly and ethically to avoid spreading misinformation or engaging in harmful activities. As artificial intelligence is developing fast, deepfake generation algorithms are simultaneously becoming more sophisticated, convincing, and hard to detect.

## Deepfake Detection

The diverse clues and detection models exploited to achieve the task of classifying fake media from genuine ones.  it will delve into the various state-of-the-art

deep learning architectures implemented in deepfake detection techniques and provide a summary of several recent deepfake detection models.

## Deepfake Detection Clues

Deepfakes can be detected by exploiting various clues, as summarized in **Figure 2**. One approach is to analyse spatial inconsistencies by closely examining deepfakes for visual artifacts, facial landmarks, or intra-frame inconsistencies. Another method involves detecting convolutional traces that are often present in deepfakes as a result of the generation process, for instance, bi-granularity artifacts and GAN fingerprints. Additionally, biological signals such as abnormal eye blinking frequency, eye colour, and heartbeat can also indicate the presence of a deepfake, as can temporal inconsistencies or the discontinuity between adjacent video frames, which may result in flickering, jittering, and changes in facial position. Poor alignment of facial emotions on swapped faces in deepfakes is a high-level semantic feature used in detection techniques. Detecting audio-visual inconsistencies is a multimodal approach that can be used for deepfakes that involve swapping both faces and audio. Another multimodal approach is to exploit spatial-temporal features by inspecting visual irregularities within individual video frames (intra-frame inspection) and analyzing temporal characteristics across video streams (inter-frame examination).

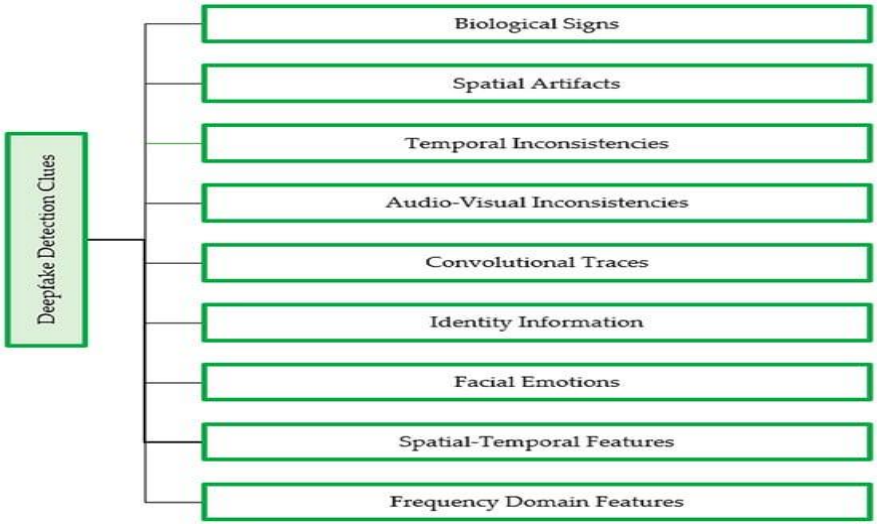



**Figure 2.** Clues and features employed by deepfake detection models in the identification of deepfake content.

## Detection Based on Spatial Artifacts

To effectively use face landmark information, in Ref., Liang et al. described a facial geometry prior module. The model harnesses facial maps and correlation within the frequency domain to study the distinguishing traits of altered and unmanipulated regions by employing a CNN-LSTM network. In order to predict manipulation localization, a decoder is utilized to acquire the mapping from low-resolution feature maps to pixel-level details, and SoftMax function was implemented for the classification task. A different approach, dubbed forensic symmetry, by Li, G. et al., assessed whether the natural features of a pair of mirrored facial regions are identical or dissimilar. The symmetry attribute extracted from frontal facial images and the resemblance feature obtained from profiles of the face images are obtained by a multi-

stream learning structure that uses DRN as its backbone network. The difference between the two symmetrical face patches is then quantified by mapping them into angular hyperspace. A heuristic prediction technique was used to put this model into functioning at the video level. As a further step, a multi-margin angular loss function was developed for classification.

Hu et al. proposed DeepfakeMAE which is a detection model that can leverage the commonalities across all facial components. To be more specific, a masked autoencoder is pretrained to learn facial part consistency by randomly masking some facial features and rebuilding missing sections using the facial parts that are still visible. This is performed given a real face image. Moreover, a model employing two networks, both utilizing pre-trained encoders and decoders, is leveraged to optimize the differentiation between authentic and counterfeit videos. Yang, J. et al. tackled deepfake detection from a different perspective where they simulate the fake image generation process to explore forgery traces. A multi-scale self-texture attention Generative Network is suggested for this aim employing an encoder–decoder generator, Resnet as backbone network, and the self-texture attention method to improve the texture characteristics in the process of disassembling an image. Additionally, a loss function termed Prob-tuple loss confined by classification probability is suggested. To identify visual artifacts at different scales, Wang et al. introduced a Multi-modal Multi-scale Transformer that works on patches of various sizes to identify disparities within images at various spatial tiers as well as forgery artifacts in the frequency domain; and the latter is added to RGB information by means of a cross modality fusion block. An approach based on GANs for deepfake detection is suggested by Xiao et al., leveraging the concealed gradient data within the grayscale representation of the manipulated image and incorporating focal loss for the classification task.

### Detection Based on Biological/Physiological Signs

Li, Y. et al. adopted an approach based on identifying eye blinking, a biological signal that is not easily conveyed in deepfake videos. Therefore, a deepfake video can be identified by the absence of eye blinking. To spot open and closed eye states, a deep neural network model that blends CNN and a recursive neural network is used while taking into account previous temporal knowledge. Alternatively, Hernandez-Ortega et al. present an innovative approach for detecting deepfake videos that focuses on analyzing heart rate information through remote photoplethy smography (rPPG). By examining video sequences and identifying slight alterations in skin color, the existence of human blood beneath the tissues can be revealed. The proposed detection system, called DeepfakesON-Phys, incorporates a Convolutional Attention Network to extract spatial and temporal details from video frames and effectively combine the two origins for improved fake video detection.

### Detection Based on Audio-Visual Inconsistencies

Boundary Aware Temporal Forgery Detection is a multimodal technique introduced by Cai et al. for correctly predicting the borders of fake segments based on visual and auditory input. While an audio encoder using a 2DCNN learns characteristics extracted from the audio, a video encoder leveraging a 3DCNN learns frame-level spatial-temporal information. Yang, W. et al. also exploited discrepancy between audio and visual elements for deepfake identification. A temporal-spatial

encoder for feature embedding explores the disparity between audio and visual components at temporal and spatial levels and a multi-modal joint-decoder, designed to concurrently acquire knowledge of multi-modal interactions and integrate audio-visual data, alongside the cross-modal classifier incorporated for manipulation detection. Similarly performed by considering both the audio and visual aspects of a video, Ilyas et al. introduced an end-to-end method called AVFakeNet. The detection model is comprised of a Dense Swin Transformer Net (DST-Net).

### Detection Based on Convolutional Traces

To detect deepfakes, Huang et al. harnessed the imperfection of the up-sampling process in GAN-generated deepfakes by employing a map of gray-scale fakeness. Furthermore, attention mechanism, augmentation of partial data, and clustering of individual samples are employed to improve the model's robustness. Chen et al. exploited a different trace which is bi-granularity artifacts, intrinsic-granularity artifacts that are caused by up-convolution or up-sampling operations, and extrinsic granularity artifacts that are the result of the post-processing step that blends the synthesized face to the original video. Deepfake detection is tackled as a multi-task learning problem where ResNet-18 is used as the backbone feature extractor. Whereas L. Guarnera et al. provided a method that uses an expectation maximization algorithm to extract a set of local features intended to simulate the convolutional patterns frequently found in photos. The five currently accessible architectures are GDWCT, StarGAN, AttGAN, StyleGAN, and StyleGAN2. Next, naive classifiers are trained to differentiate between real images and those produced by these designs.

### Detection Based on Identity Information

Based on the intuition that every person can exhibit distinct patterns in the simultaneous occurrence of their speech, facial expressions, and gestures, Agarwal et al. introduced a multimodal detection method with a semantic focus that incorporates speech transcripts into gestures specific to individuals analysis using interpretable action units to model facial and cranial motion of an individual. Meanwhile, Dong et al. proposed an Identity Consistency Transformer that learns simultaneously and identifies vectors for the inner face and another for the outer face; moreover, the model uses a novel consistency loss to drive both identities apart when their labels are different and to bring them closer when their labels are the same. Similarly, Nirkin et al. identified deepfakes by looking for identity-to-identify inaccuracies between two identity vectors that represent the inner face region and its outer context. The identity vectors are obtained using two networks based on the Xception architecture and trained using a vanilla cross entropy loss. Focusing on temporal identity inconsistency, Liu et al. introduced a model that captures the disparities of faces within video frames of the same person by encoding identity information in all frames to identity vectors and learning from these vectors the temporal embeddings, thus identifying inconsistencies. The proposed model integrates triplet loss for enhanced discrimination in learning temporal embeddings.

### Detection Based on Facial Emotions

Despite the fact that deepfakes can produce convincing audio and video, it can be difficult to produce material that maintains coherence concerning high-level

semantics, including emotions. Unnatural displays of emotion, as determined by characteristics like valence and arousal, where arousal indicates either heightened excitement or tranquility and valence represents positivity or negativity of the emotional state, can offer compelling proof that a video has been artificially created. Using the emotion inferred from the visage and vocalizations of the speaker, Hosler et al. introduced an approach for identifying deepfakes. The suggested method makes use of long, short-term memory networks and visual descriptors to infer emotion from low-level audio emotion; a supervised classifier is then incorporated to categorize videos as real or fake using the predicted emotion. Leveraging the same high-level features, Conti et al. focused on identifying deepfake speech tracks created using text-to-speech (TTS) algorithms that manipulate the emotional tone of the voice content. To extract emotional features, a Speech Emotion Recognition network trained on a speech dataset labeled with the speaker's emotional expression is employed, alongside a supervised classifier that receives emotional features as input and predicts the authenticity of the provided speech track as either genuine or deepfake.

### Detection Based on Temporal Inconsistencies

To leverage temporal coherence to detect deepfakes, Zheng et al. proposed an approach to reduce the spatial convolution kernel size to 1 while keeping the temporal convolution kernel size constant using a fully temporal convolution network in addition to a Transformer Network that explores the long-term temporal coherence. Pei et al. exploited the temporal information in videos by incorporating a Bidirectional-LSTM model. Gu et al. proposed a Region-Aware Temporal Filter module to generate temporal filters to distinct spatial areas by breaking down the dynamic temporal kernel into fundamental, region-independent filters. Additionally, region-specific aggregation weights are introduced to steer these regions in adaptively acquiring knowledge of temporal incongruities. The input video is split into multiple snippets to cover the long-term temporal dynamics. Inspired by how humans detect fake media through browsing and scrutinizing, Ru et al. presented a model dubbed Bita-Net which consists of two pathways: one that checks the temporal consistency by rapidly scanning the entire video, and a second pathway improved by an attention branch to analyze key frames of the video at a lower rate.

### Detection Based on Spatial-Temporal Features

The forced mixing of the manipulated face in the generation process of deepfakes causes spatial distortions and temporal inconsistencies in crucial facial regions, which Sun et al. proposed to reveal by extracting the displacement trajectory of the facial region. For the purpose of detecting fake trajectories, a fake trajectory detection network, utilizing a gated recurrent unit backbone in conjunction with a dual-stream spatial-temporal graph attention mechanism, is created. In order to detect the spatial-temporal abnormalities in the altered video trajectory, the network makes use of the extracted trajectory and explicitly integrates the important data from the input sequences. Lu et al. proposed a detection method based on an improved Capsule Network and the fusion of temporal–spatial features. The optical flow algorithm effectively captures the temporal characteristics of manipulated videos, and the improved Capsule Network reaches a thorough conclusion by considering temporal–spatial features using weight initialization and updating on a dynamic routing algorithm. Meanwhile, Waseem et al. described a dual-stream convolutional neural network

strategy is employed, incorporating XceptionNet and 3DCNN, to capture spatial irregularities and temporal variations. Initially, MTCNN is employed for face detection and extraction from input video frames. Subsequently, 3DCNN and XceptionNet are utilized to extract features from facial images. Finally, fully connected layers and sigmoid layers determine the authenticity of the video.

## Deep Learning Models for Deepfake Detection

Several advanced technologies have been employed in the domain of deepfake detection, such as machine learning and media forensics-based approaches. However, it is widely acknowledged that deep learning-based models currently exhibit the most remarkable performance in discerning between fabricated and authentic digital media. These models leverage sophisticated neural network architectures known as backbone networks, displayed in **Figure 3**, which have demonstrated exceptional efficacy in computer vision tasks. Prominent examples of such architectures include VGG, EfficientNet, Inception, CapsNet, and ViT, and are particularly renowned for their prowess in the feature extraction phase. Deep learning-based detection models go beyond conventional methods by incorporating additional techniques to further enhance their performance. One such approach is meta-learning, which enables the model to learn from previous experiences and adapt its detection capabilities accordingly. By leveraging meta-learning, these models become more proficient at recognizing patterns and distinguishing between genuine and manipulated content.

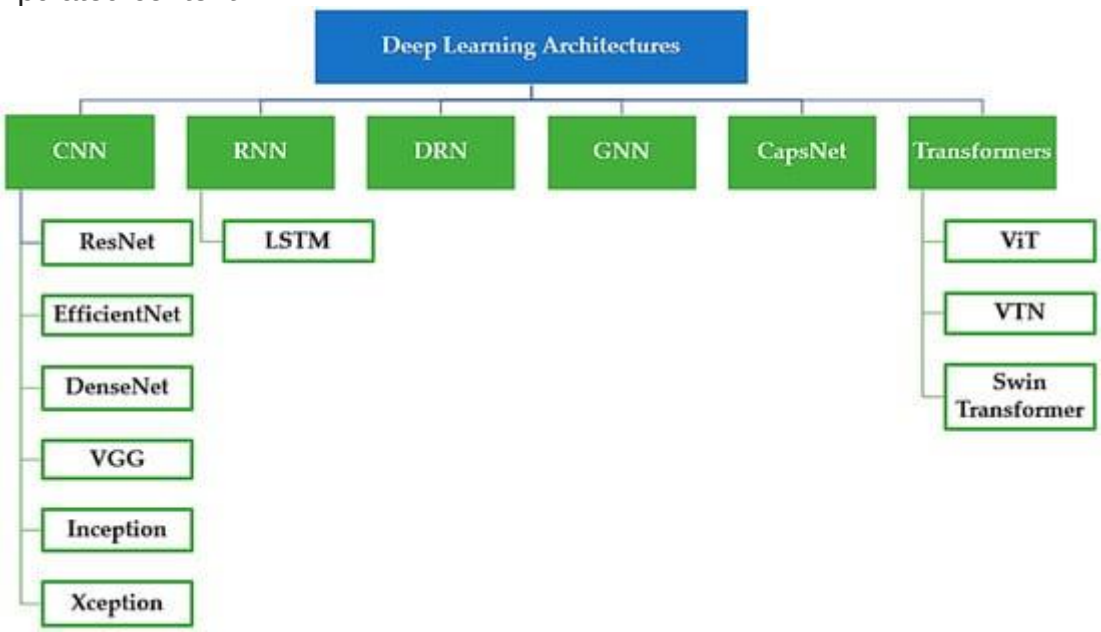



**Figure 3.** Overview of predominant deep learning architectures, networks, and frameworks employed in the development of deepfake detection models.

Furthermore, data augmentation plays a crucial role in training deep learning-based detection models. This technique involves augmenting the training dataset with synthetic or modified samples, which enhances the model's capacity to generalize and recognize diverse variations of deepfake media. Data augmentation enables the model to learn from a wider range of examples and improves its robustness against different types of manipulations. Attention mechanisms have also proven to be valuable additions to deep learning-based detection models. By directing the model's

focus toward relevant features and regions of the input data, attention mechanisms enhance the model's discriminative power and improve its overall accuracy. These mechanisms help the model select critical details, making it more effective in distinguishing between real and fake media. Collectively, the combination of deep learning-based architectures, meta-learning, data augmentation, and attention mechanisms has significantly advanced the field of deepfake detection. These technologies work in harmony to equip models with the ability to identify and flag manipulated media with unprecedented accuracy.

The Convolutional Neural Network is a powerful deep learning algorithm designed for image recognition and processing tasks. It consists of various levels, encompassing convolutional layers, pooling layers, and fully connected layers. There are different types of CNN models used in deepfake detection such as ResNet, short for Residual Network, which is an architecture that introduces skip connections to fix the vanishing gradient problem that occurs when the gradient diminishes significantly during backpropagation; these connections involve stacking identity mappings and skipping them, utilizing the layer's prior activations. This technique accelerates first training by reducing the number of layers in the network. The concept underlying this network is different from having the layers learn the fundamental mapping. Rather than directly defining the initial mapping as H(x), we let the network adapt and determine it, as shown in **Figure 4**.

$$F(x) := H(x) - x \text{ which gives } H(x) := F(x) + x.$$

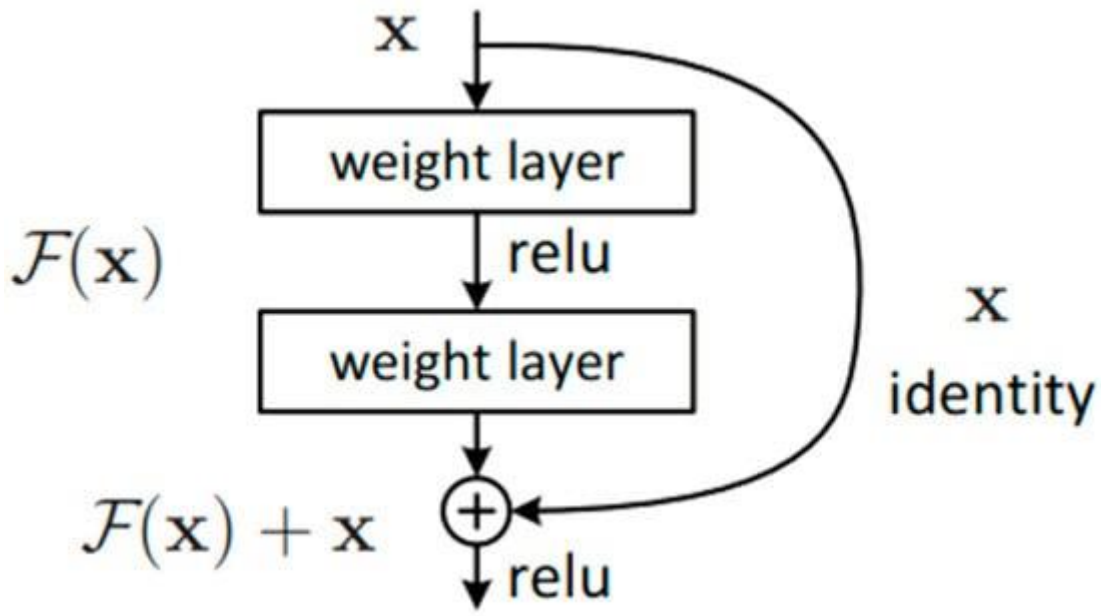



**Figure 4.** ResNet building block (source:).

Another architecture based on CNNs is VGG, short for Visual Geometry Group, which comprises multiple layers. Instead of using large kernel sized filters, this architecture utilizes multiple filters with a kernel size of 3 × 3. The VGG16 architecture employs a doubling of filters at each convolutional layer, a fundamental design principle. However, a notable drawback of the VGG16 network is its substantial size, resulting in extended training times due to its depth and numerous fully connected layers. The model's file size exceeds 533 MB, rendering the implementation of a VGG network a time-intensive endeavour.

An additional significant CNN-based architecture in deepfake detection models is EfficientNet. It has a scaling method that applies a uniform scaling approach to all dimensions of depth, width, and resolution. This is achieved by utilizing a compound coefficient. In **Figure 5**, the performance of EfficientNet is presented alongside other network architectures. The largest model within the EfficientNet series, EfficientNet B7, achieved remarkable results on both the ImageNet and CIFAR-100 datasets. Specifically, it achieved approximately 84.4% in top-1 accuracy and 97.3% in top-5 accuracy on the ImageNet dataset. Furthermore, this model was not only significantly more compact, being 8.4 times smaller, but also notably faster, with a speedup of 6.1 times compared to the prior leading CNN model. Additionally, it exhibited strong performance with 91.7% accuracy on the CIFAR-100 dataset and an impressive 98.8% accuracy on the Flowers dataset.

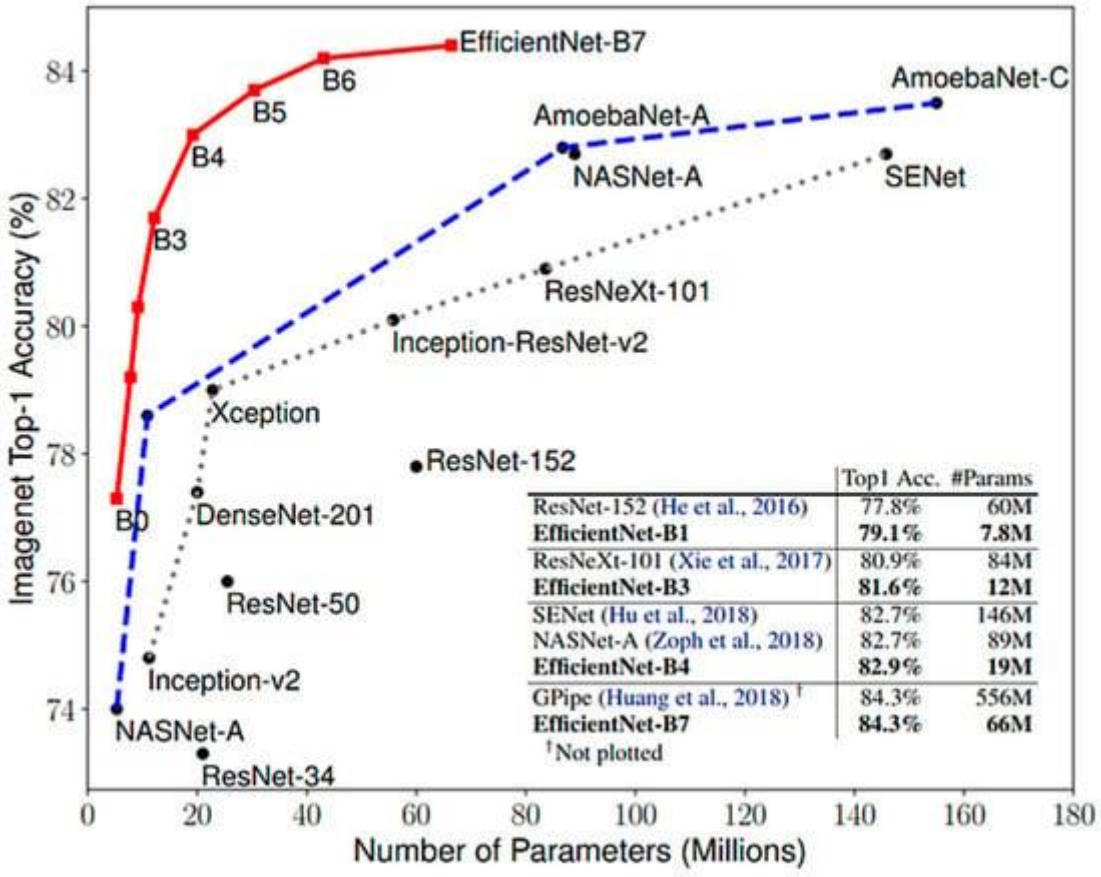



**Figure 5.** EfficientNet performance on the ImageNet dataset (source:).

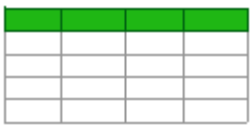
Inception models help mitigate the computational cost and other overfitting in CNN architectures by utilizing stacked 1 × 1 convolutions for dimensionality reduction. Xception, developed by researchers at Google, is an advanced version of the Inception architecture. It offers a novel approach by reinterpreting Inception modules as an intermediate step between standard convolution and depthwise separable convolution. While the conventional convolution operation combines channel-wise and spatial-wise computations in a single step, depthwise separable convolution divides this process into two distinct steps. Firstly, it employs depthwise convolution to apply an individual convolutional filter to each input channel, and subsequently, pointwise convolution is employed to create a linear combination of the results obtained from the depthwise convolution.

An alternative to CNNs would be Capsule Networks that are able to retrieve spatial information as well as other important details to avoid the information loss seen during pooling operations. Capsules exhibit equivariance characteristics and consist of a neural network that handles vectors as inputs and outputs, in contrast to the scalar values processed by CNNs. This unique attribute of capsules enables them to capture not only the features of an image, but also its deformations and various viewing conditions. Within a capsule network, each capsule comprises a cluster of neurons, with each neuron's output signifying a distinct attribute of the same feature. This structure offers the advantage of recognizing the entire entity by first identifying its constituent parts.

Recurrent Neural Networks are a kind of neural network that handles sequential data by feeding it in a sequential manner. They are specifically designed to tackle the challenge of time-series data, where the input is a sequence of data points. In an RNN, the input not only includes the current data point but also the previous ones. This creates a directed graph structure between the nodes, following the temporal sequence of the data. Additionally, each neuron in an RNN has its own internal memory, which retains information from the computations performed on the previous data points. LSTM, or Long Short-Term Memory, is a specific type of recurrent neural network that addresses the challenge of long-term dependencies in sequential data by allowing more accurate predictions based on recent information. While traditional RNNs struggle as the gap between relevant information increases, LSTM networks excel at retaining information over extended periods. This capability makes LSTM particularly effective for processing, predicting, and classifying time-series data.

A new model that has emerged as a strong alternative to convolutional neural networks is the vision transformer. ViT models exhibit exceptional performance, surpassing the state-of-the-art CNNs by nearly four times in both computational efficiency and accuracy. Transformers, which are non-sequential deep learning models, play a significant role in vision transformers. They utilize the self-attention mechanism, assigning varying degrees of importance to different segments of the input data. The Swin Transformer is a type of ViTs that exhibits versatility in modeling at different scales and maintains linear computational complexity concerning image size. This advantageous combination of features enables the Swin Transformer to be well suited for a wide array of vision tasks, encompassing image classification, object detection, and semantic segmentation, among others. Another variant of transformers is Video Transformers, which are efficient for evaluating videos on a large scale, ensuring optimal utilization of computational resources and reduced wall runtime. This capability enables full video processing during test time, making VTNs particularly well-suited for handling lengthy videos. **Table 2** shows some of the recent detection techniques.

**Table 2.** Summary of recent deepfake detection models, employed techniques, feature sets, datasets, and intra-dataset performance results.



## Datasets

In the context of deepfakes, datasets serve as the foundation for training, testing, and benchmarking deep learning models. The accessibility of reliable and diverse datasets plays a crucial role in the development and evaluation of deepfake techniques. A variety of important datasets, summarized in **Table 3**, have been curated specifically for deepfake research, each addressing different aspects of the problem and contributing to the advancement of the field. **Figure 6** shows some of the widely used datasets in deepfake detection models' improvement.

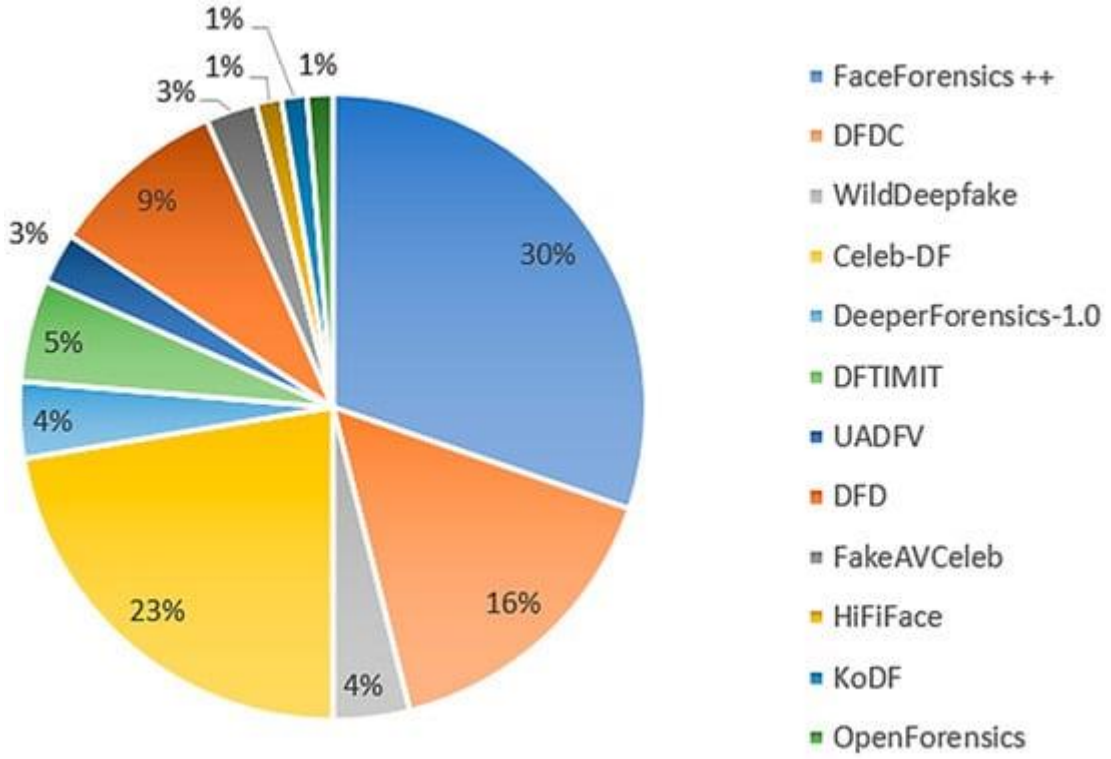



**Figure 6.** Frequency of usage of different deepfake datasets in the discussed detection models within this survey.

**Table 3.** Key characteristics of the most prominent and publicly available deepfake datasets.

| | | | |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |

FaceForensics++ is a well-known dataset used for deepfake detection that was provided in 2019 as an addition to the FaceForensics dataset, which was made available in 2018 and only included videos with altered facial expressions. Four subsets of the FF++ dataset are available: FaceSwap, Deepfake, Face2Face, and NeuralTextures. It includes 3000 edited videos in addition to 1000 original videos that were pulled from the YouTube-8M dataset. The dataset can be used to test deepfake detection strategies on both compressed and uncompressed videos because it is supplied in two different quality levels. The FF++ dataset has limits when it comes to spotting lip-sync deepfakes, and some videos might have color discrepancies near the modified faces.

DFDC, the deepfake detection challenge dataset hosted by Facebook, stands as the most extensive collection of face swap videos available and openly accessible. It

contains over 100,000 total clips sourced from 3426 paid actors from diverse backgrounds, including different genders, ages, and ethnic groups.

DeeperForensics-1.0 is a significant dataset available for detecting deepfakes that contains 50,000 original clips and 10,000 forged ones. These manipulated videos were generated using a conditional autoencoder called DF-VAE. The dataset includes a broad range of actor appearances and is designed to represent real-world scenarios more accurately by including a blend of alterations and disturbances, including compression, blurriness, noise, and other visual anomalies.

WildDeepfake is a dataset that is widely recognized as a difficult one for deepfake detection. It features both authentic and deepfake samples obtained from the internet, which distinguishes it from other available datasets. While previous datasets have only included synthesized facial images, this dataset includes a variety of body types. However, there remains a need for a more comprehensive dataset that can generate full-body deepfakes to improve the robustness of deepfake detection models.

Celeb-DF dataset is a collection of authentic and synthesized deepfake videos that are visually similar in quality to those that are commonly shared online. This dataset represents a significant expansion of its first version, which contained only 795 deepfake videos. Celeb-DF comprises 590 unaltered videos sourced from YouTube, featuring individuals of varying ages, ethnicities, and genders, along with 5639 associated deepfake videos, all of which were created using readily accessible YouTube excerpts featuring 59 famous personalities from diverse backgrounds. The deepfake videos were generated using an advanced synthesis method, resulting in more realistic and convincing deepfakes.

Finding fake faces among numerous genuine faces in scenes taken in the nature is a significant difficulty. OpenForensics dataset was specifically created with face-wise rich annotations for the detection and segmentation of face forgeries. The OpenForensics dataset has a lot of potential for study in generic human face detection and deepfake prevention because of its extensive annotations. A total of 334 K human faces are depicted among 115 K photos in version 1.0.0. This collection includes numerous individuals with different origins, ages, genders, stances, positions, and face occlusions.

FakeAVCeleb is a multimodal deepfake detection dataset that includes deepfake videos and cloned deepfake audio. It features diverse celebrities in terms of ethnicity, age, and gender balance. The dataset was evaluated using 11 different deepfake detection methods, including unimodal, ensemble-based, and multimodal approaches. To create deepfake videos, 500 real videos were used as sources and generated around 20,000 deepfake videos using various techniques like face-swapping and facial reenactment.

DeepfakeTIMIT is a dataset containing 620 videos where faces were swapped using GAN-based techniques. It was created by selecting 16 pairs of similar-looking individuals from the VidTIMIT database, with two quality levels for each pair (64 × 64 and 128 × 128). The original audio tracks were retained without any alterations.

UADFV dataset includes 98 videos, totaling 32,752 frames, evenly split between 49 real videos and 49 fake ones. Each video features a single subject and lasts around 11 s. Among these videos, there are 49 original real videos, which were manipulated to generate 49 Deep Fake videos.

DFD or DeepFakeDetection is a dataset created by Google and Jigsaw and encompasses a wide range of scenes that consist of more than 363 genuine sequences featuring 28 paid actors across 16 different scenes. Additionally, it includes over 3000 manipulated videos.

**HiFiFace** is a dataset that contains 1000 fake videos from FaceForensics++, meticulously adhering to the source and target pair configurations defined in FF++. Additionally, it includes 10,000 frames extracted from FF++ videos, facilitating quantitative testing.

**KoDF** is an extensive compilation of synthesized and authentic videos primarily centered around Korean subjects. Its primary objective is to support the advancement of deepfake detection methods. This dataset comprises 62,166 authentic videos and 175,776 fake videos, featuring 403 different subjects.

One of the challenges faced by researchers in the field of deepfakes is the lack of comprehensive and diverse datasets for deepfake detection. Existing datasets either have limited diversity, meaning they do not cover a wide range of scenarios and variations, or only focus on basic forgery detection without capturing the intricacies and subtleties of advanced deepfakes. To address this problem and push the boundaries of deepfake detection, researchers and technology companies have taken up the task of constructing several benchmarks. These benchmarks serve as standardized datasets that encompass a broad range of facial variations, lighting conditions, camera angles, and other relevant factors. By including diverse samples, these benchmarks enable researchers to develop and evaluate advanced algorithms and techniques for detecting and analyzing deepfakes more effectively. To mention a few, **ForgeryNet** is an extremely large deepfake benchmark with consistent annotations in both image and video data for four distinct tasks: Image Forgery Classification, Spatial Forgery Localization, Video Forgery Classification, and Temporal Forgery Localization. It consists of 2.9 million images, 221,247 videos and 15 manipulation methods.

For the predominant focus on a single modality and limited coverage of forgery methods, current datasets for deepfake detection are primarily constrained when it comes to audio-visual deepfakes. **DefakeAVMiT** is a dataset includes an ample amount of deepfake visuals paired with corresponding audios and generated by various deepfake methods affecting either modality. Alternatively, **LAV-DF** consists of content-driven manipulations to help with the detection of content altering fake segments in videos due to the lack of suitable datasets for this task. It is important to note that the availability and creation of datasets are ongoing processes, with new datasets being introduced and existing ones being expanded or refined over time. The continuous development of diverse and representative datasets is crucial to ensure the robustness and generalizability of deepfake detection algorithms, as well as to keep up with the evolving techniques employed by malicious actors.

Dataset preprocessing plays a crucial role in training a Deep Fake detection model. Preprocessing techniques help in enhancing the quality and effectiveness of the dataset, improving the performance of the detection model.

Here are some key aspects highlighting the importance of dataset preprocessing and potential preprocessing techniques for Deep Fake detection:

1. Data Cleaning: Data cleaning involves removing any corrupt or irrelevant data from the dataset. In Deep Fake detection, this step helps ensure that the dataset consists of high-quality and reliable samples. It may involve removing duplicates, irrelevant frames, or videos with low resolution or quality.

2. Labeling and Annotation: Proper labelling and annotation are essential for training a Deep Fake detection model. Each sample in the dataset needs to be accurately labelled as either genuine or manipulated. Additionally, annotating specific

regions of interest, such as facial landmarks or manipulated areas, can aid in training the model to focus on relevant features.

3. Frame Extraction and Selection: Deep Fake videos often consist of multiple frames or frames with different levels of manipulation. Extracting and selecting representative frames from each video can help create a diverse and balanced dataset. This ensures that the model learns from various visual cues and can generalize well to different Deep Fake scenarios.

4. Augmentation Techniques: Data augmentation techniques can be applied to increase the dataset size and improve the model's robustness. Augmentation methods such as rotation, flipping, scaling, and brightness adjustment can help create additional variations of each sample in the dataset. This can assist in reducing over fitting and improving the model's ability to detect Deep Fakes in real-world scenarios.

5. Bias Correction: It is essential to address any biases in the dataset during preprocessing. Biases can arise due to differences in distribution or characteristics between genuine and manipulated videos. Correcting these biases can help ensure a fair and unbiased training process, leading to a more accurate and reliable detection model.

6. Normalization and Standardization: Normalization and standardization techniques can be applied to bring the features of the dataset to a similar scale and distribution. This step helps in improving the convergence and stability of the model during training. Normalizing pixel values, applying z-score scaling, or using other normalization techniques can be beneficial for Deep Fake detection.

7. Dataset Balancing: Ensuring a balanced dataset is important to prevent the model from favouring one class over the other. Deep Fake detection datasets often have an imbalance in the number of genuine and manipulated videos. Techniques such as oversampling the minority class or under sampling the majority class can help achieve a balanced dataset for more effective training.

By implementing appropriate dataset preprocessing techniques, the Deep Fake detection model can learn from a high-quality, diverse, and balanced dataset. This enhances the model's ability to generalize well, detect various Deep Fake manipulation techniques, and adapt to real-world scenarios.

One of the key measures to mitigate the negative impacts of Deep Fakes is the development and implementation of robust detection mechanisms.

These detection mechanisms play a crucial role in identifying and preventing the spread of manipulated media. Here are some important aspects to consider:

1. Advancing Technology: Continuous research and development of advanced detection algorithms are essential to keep pace with evolving Deep Fake techniques. By leveraging machine learning, computer vision, and natural language processing, these mechanisms can analyse various visual and audio cues to identify discrepancies or anomalies that indicate manipulation.

2. Collaborative Efforts: Addressing Deep Fake concerns requires collaboration between technology developers, researchers, policymakers, and social media platforms. Sharing knowledge, resources, and best practices can enhance the effectiveness of detection mechanisms and enable a collective response to combat the spread of malicious Deep Fakes.

3. Data Sharing and Labeling: Establishing comprehensive datasets of known Deep Fakes, combined with accurate labelling, can facilitate the training and evaluation of detection models. This data can be sourced from both synthetic and real-world Deep Fake examples, enabling the algorithms to learn patterns and characteristics specific to manipulated content.

4. User Education and Awareness: Educating the public about Deep Fakes is crucial to empower individuals in identifying and critically assessing the authenticity of media they encounter. Promoting media literacy, teaching digital media literacy skills, and creating awareness campaigns can help users become more skeptical and discerning consumers of content.

5. Platform Responsibility: Social media platforms and content-sharing platforms have a significant role in preventing the spread of Deep Fakes. Implementing policies and guidelines to regulate the sharing and promotion of manipulated content, as well as investing in automated detection systems, can act as an effective deterrent.

6. Legal and Policy Frameworks: Governments and legal authorities need to adapt and enforce legislation that addresses Deep Fake creation and dissemination. Clear guidelines, regulations, and consequences for producing and sharing malicious Deep Fakes can foster an environment of accountability and deterrence. In conclusion, detection mechanisms are vital in addressing the ethical implications of Deep Fake technology. By combining technological advancements, collaborative efforts, user education, platform responsibility, and legal frameworks, we can strive towards minimizing the harmful impact of Deep Fakes while safeguarding trust and integrity in digital content.

## Coding for deepfake detection in kaggle using python

### Coding part for video in googlecolab extracting frames from video

```
import cv2
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
video_path='/content/drive/My Drive/file folder/file.mp4'
cap=cv2.VideoCapture(video_path)
if not cap.isOpened():
  print("Video File Not Loaded. Retry with a different")
(create folder in my drive file folder)
else:
  frames_dir='/content/drive/My Drive/ file folder /video_frames'
  frame_count=0
  while True:
    ret, frame = cap.read() #Meta Information & The acutal Frame
    if not ret:
      break
    frame_count +=1
    frame_name=f'frame_{frame_count}.jpg'
    frame_path=f'{frames_dir}/{frame_name}'
    cv2.imwrite(frame_path, frame)
  print(f'Total Number of frames extracted are {frame_count}')
cap.release()
```

### After video converted into image frames deepfake detection code for images performed in kaggle using import the dataset from extracted frames

```
!pip install -U --upgrade tensorflow
import sys
import sklearn
import tensorflow as tf
```

```
import cv2
import pandas as pd
import numpy as np

import plotly.graph_objs as go
from plotly.offline import iplot
from matplotlib import pyplot as plt
import matplotlib.pyplot as plt

plt.rc('font', size=14)
plt.rc('axes', labelsize=14, titlesize=14)
plt.rc('legend', fontsize=14)
plt.rc('xtick', labelsize=10)
plt.rc('ytick', labelsize=10)
import os

def get_data():
    return pd.read_csv('../input/deepfake-faces/metadata.csv')
meta=get_data()
meta.head()
meta.shape

len(meta[meta.label=='FAKE']),len(meta[meta.label=='REAL'])
real_df = meta[meta["label"] == "REAL"]
fake_df = meta[meta["label"] == "FAKE"]
sample_size = 400

real_df = real_df.sample(sample_size, random_state=42)
fake_df = fake_df.sample(sample_size, random_state=42)

sample_meta = pd.concat([real_df, fake_df])
from sklearn.model_selection import train_test_split

Train_set, Test_set =
train_test_split(sample_meta,test_size=0.2,random_state=42,stratify=sample_
meta['label'])
Train_set, Val_set  =
train_test_split(Train_set,test_size=0.3,random_state=42,stratify=Train_set['lab
el'])
Train_set.shape,Val_set.shape,Test_set.shape
y = dict()

y[0] = []
y[1] = []

for set_name in (np.array(Train_set['label']), np.array(Val_set['label']),
np.array(Test_set['label'])):
    y[0].append(np.sum(set_name == 'REAL'))
```

```
    y[1].append(np.sum(set_name == 'FAKE'))

trace0 = go.Bar(
    x=['Train Set', 'Validation Set', 'Test Set'],
    y=y[0],
    name='REAL',
    marker=dict(color='#33cc33'),
    opacity=0.7
)

trace1 = go.Bar(
    x=['Train Set', 'Validation Set', 'Test Set'],
    y=y[1],
    name='FAKE',
    marker=dict(color='#ff3300'),
    opacity=0.7
)

data = [trace0, trace1]

layout = go.Layout(
    title='Count of classes in each set',
    xaxis={'title': 'Set'},
    yaxis={'title': 'Count'}
)

fig = go.Figure(data, layout)
iplot(fig)
plt.figure(figsize=(15,15))

for cur,i in enumerate(Train_set.index[25:50]):
    plt.subplot(5,5,cur+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)

    plt.imshow(cv2.imread('../input/deepfake-
faces/faces_224/'+Train_set.loc[i,'videoname'][:-4]+'.jpg'))

    if(Train_set.loc[i,'label']=='FAKE'):
        plt.xlabel('FAKE Image')
    else:
        plt.xlabel('REAL Image')

plt.show()

def retreive_dataset(set_name):
    images,labels=[],[]
```

```
    for (img, imclass) in zip(set_name['videoname'], set_name['label']):
        images.append(cv2.imread('../input/deepfake-faces/faces_224/'+img[:-
4]+'.jpg'))
        if(imclass=='FAKE'):
            labels.append(1)
        else:
            labels.append(0)

    return np.array(images),np.array(labels)

X_train,y_train=retreive_dataset(Train_set)
X_val,y_val=retreive_dataset(Val_set)
X_test,y_test=retreive_dataset(Test_set)

from functools import partial

tf.random.set_seed(42)

DefaultConv2D = partial(tf.keras.layers.Conv2D, kernel_size=3,
padding="same",
                        activation="relu", kernel_initializer="he_normal")

model = tf.keras.Sequential([
    DefaultConv2D(filters=64, kernel_size=7, input_shape=[224, 224, 3]),
    tf.keras.layers.MaxPool2D(),
    DefaultConv2D(filters=128),
    DefaultConv2D(filters=128),
    tf.keras.layers.MaxPool2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=128, activation="relu",
                          kernel_initializer="he_normal"),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(units=64, activation="relu",
                          kernel_initializer="he_normal"),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(units=1, activation="sigmoid")
])

model.compile(loss="binary_crossentropy", optimizer="nadam",
              metrics=["accuracy"])
model.summary()

history = model.fit(X_train, y_train, epochs=5,batch_size=64,
                    validation_data=(X_val, y_val))

score = model.evaluate(X_test, y_test)

# plot model performance
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(1, len(history.epoch) + 1)

plt.figure(figsize=(15,5))

plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Train Set')
plt.plot(epochs_range, val_acc, label='Val Set')
plt.legend(loc="best")
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Train Set')
plt.plot(epochs_range, val_loss, label='Val Set')
plt.legend(loc="best")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Model Loss')

plt.tight_layout()
plt.show()

train_set_raw=tf.data.Dataset.from_tensor_slices((X_train,y_train))
valid_set_raw=tf.data.Dataset.from_tensor_slices((X_val,y_val))
test_set_raw=tf.data.Dataset.from_tensor_slices((X_test,y_test))

tf.keras.backend.clear_session()  # extra code – resets layer name counter

batch_size = 32
preprocess = tf.keras.applications.xception.preprocess_input
# Find out the most accurate images for the sequence
train_set = train_set_raw.map(lambda X, y: (preprocess(tf.cast(X, tf.float32)),
y))
train_set = train_set.shuffle(1000, seed=42).batch(batch_size).prefetch(1)
valid_set = valid_set_raw.map(lambda X, y: (preprocess(tf.cast(X, tf.float32)),
y)).batch(batch_size)
test_set = test_set_raw.map(lambda X, y: (preprocess(tf.cast(X, tf.float32)),
y)).batch(batch_size)

# extra code – displays the first 9 images in the first batch of valid_set
plt.figure(figsize=(12, 12))
for X_batch, y_batch in valid_set.take(1):
    for index in range(9):
```

```
        plt.subplot(3, 3, index + 1)
        plt.imshow((X_batch[index] + 1) / 2)  # rescale to 0–1 for imshow()
        if(y_batch[index]==1):
            classt='FAKE'
        else:
            classt='REAL'
        plt.title(f"Class: {classt}")
        plt.axis("off")

plt.show()

data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip(mode="horizontal", seed=42),
    tf.keras.layers.RandomRotation(factor=0.05, seed=42),
    tf.keras.layers.RandomContrast(factor=0.2, seed=42)
])

# extra code – displays the same first 9 images, after augmentation
plt.figure(figsize=(12, 12))
for X_batch, y_batch in valid_set.take(1):
    X_batch_augmented = data_augmentation(X_batch, training=True)
    for index in range(9):
        plt.subplot(3, 3, index + 1)
        # We must rescale the images to the 0-1 range for imshow(), and also
        # clip the result to that range, because data augmentation may
        # make some values go out of bounds (e.g., RandomContrast in this
case).
        plt.imshow(np.clip((X_batch_augmented[index] + 1) / 2, 0, 1))
        if(y_batch[index]==1):
            classt='FAKE'
        else:
            classt='REAL'
        plt.title(f"Class: {classt}")
        plt.axis("off")

plt.show()

tf.random.set_seed(42)  # extra code – ensures reproducibility
base_model = tf.keras.applications.xception.Xception(weights="imagenet",
                                                     include_top=False)
avg = tf.keras.layers.GlobalAveragePooling2D()(base_model.output)
output = tf.keras.layers.Dense(1, activation="sigmoid")(avg)
model = tf.keras.Model(inputs=base_model.input, outputs=output)

for layer in base_model.layers:
    layer.trainable = False

optimizer = tf.keras.optimizers.SGD(learning_rate=0.1, momentum=0.9)
```

```
model.compile(loss="binary_crossentropy", optimizer=optimizer,
              metrics=["accuracy"])
history = model.fit(train_set, validation_data=valid_set, epochs=3)

for indices in zip(range(33), range(33, 66), range(66, 99), range(99, 132)):
    for idx in indices:
        print(f"{idx:3}: {base_model.layers[idx].name:22}", end="")
    print()

model.evaluate(test_set)

for layer in base_model.layers[56:]:
    layer.trainable = True

optimizer = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
model.compile(loss="binary_crossentropy", optimizer=optimizer,
              metrics=["accuracy"])
history = model.fit(train_set, validation_data=valid_set, epochs=10)

# plot model performance
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(1, len(history.epoch) + 1)

plt.figure(figsize=(15,5))

plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Train Set')
plt.plot(epochs_range, val_acc, label='Val Set')
plt.legend(loc="best")
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Train Set')
plt.plot(epochs_range, val_loss, label='Val Set')
plt.legend(loc="best")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Model Loss')

plt.tight_layout()
plt.show()

train_set_raw=tf.data.dataset.from_tensor_slices((x_train,y_train))
```

```
valid_set_raw=tf.data.dataset.from_tensor_slices((x_val,y_val))
test_set_raw=tf.data.dataset.from_tensor_slices((x_test,y_test))
tf.keras.backend.clear_session() #extra code_rests layer name counter
batch_size=32
#find out the most accurate images for the sequence
preprocess=tf.keras.applications.xception.preprocess-input
train_set=train_set_raw.map(lambda X,y:(preprocess(tf.cast (X,tf.float32)) ,y)
train_set=train_set.shuffle(1000,seed=42),batch(batch_size),prefetch(1)
valid_set=valid_set_raw.map(lambda X,y:(preprocess(tf.cast
(X,tf.float32)),y).batch(batch_size)
test_set=test_set_raw.map(lambda X,y:(preprocess(tf.cast
(X,tf.float32)),y).batch(batch_size)
#extra code_displays the first 9 images in the first batch of valid_set
first batch of valid_set
plt.figure(figsize=(12,12))
    for x_batch,y_batch in valid_set.take(1):
    for index in range(9):
    plt.subpplot(3,3,index+1)
    plt.imshow((x_batch[index]+1)/2) #rescale to 0-1 for imshow()
      if(y_batch[index]==1):
        classt='FAKE'
    else:
       classt='REAL'
 plt.title(f"class:{classt}")
 plt.axis("off")
 plt show()

data_augumentation=tf.kras.sequential([tf.keras.layers.Random
Flip(mode="horizontal",seed=42),
                              tf.keras.layers.Random Rotation
Flip(factor=0.05,seed=42)
                              tf.keras.layers.Random contrast
Flip(factor=0.2,seed=42)])

  #extra code_displays the same first 9 images,after augumentation
                     plt.figure(figsize=(12,12))
for X_batch,Y_batch in valid set.take(1):
   X_batch-Y_augumented=data_augumentation(X_batch,training=True)
      for index in range(9):
         plt.subplot(3,3,index +1)
   #we must rescale the images to the 0-1 range for imshow(),and also
   # make some values go out of bound(e.g.,Random contrast in this case).
if(Y_batch[index]==1)
   classt='FAKE'
else:
   classt='REAL'
plt.title(f"class:{classt}")
plt.axis("off")
```

```
plt.show()
tf.random.set_seed(42) #extra code_ensure reproducibility
base_model=tf.keras.applications.xception(weights="imagenet",include_top=F
alse)
avg=tf.keras.layers.GlobalAverage pooling 2D()(base_model.output)
output=tf.keras.layers.Dense(1,activation="sigmoid")(avg)
model=tf.keras.Model(inputs=base_model.input,output=output)
       for layers in base_model.layers:
                    layers.trainable=False

optimizer=tf.keras.optimizers.SGD(learning_rate=0.1,momentum=0.9)
model.compile(loss="binary_crossentropy"optimizer=optimizer,metrics=["acc
uracy"])
history=model.fit(train_set,validation_data=valid_set,epochs=3)
for indices in zip(range(33)range(33,66),range(66,99),range(99,132)):
   for idx in indices:
     print (f"{idx:3}:{base_model.layers[index].name:22}",end="")
print()
model.evaluate(test_set)
for layer in base_model.layers[56:]
  layer.trainable=True
optimizer=tf.keras.optimizers.SGP(learning_rate=0.01,momentum=0.9)
model.compile(loss="binary_crossentropy"optimizer=optimizer,metrics=["acc
uracy"])
history=model.fit(train_set,validation_data=valid_set,epochs=10)
#plot model performance
acc=history.history['accuracy']
val_acc=history.history['val_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']
epochs_range=range(1,len(history.epoch)+1)
plt.figure(figsize=(15,5))

plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Train Set')
plt.plot(epochs_range, val_acc, label='Val Set')
plt.legend(loc="best")
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Train Set')
plt.plot(epochs_range, val_loss, label='Val Set')
plt.legend(loc="best")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Model Loss')
```

```
plt.tight_layout()
plt.show()

model.evaluate(test_set)
```